芯片设计的模拟验证技术分析和探讨[1]

吕涛 李华伟 李晓维

摘要：本文通过分析模拟验证技术随着设计规模发展而遇到的困境及其原因，认为基于错误模型的设计验证是有望取得突破性进展的技术方向。本文进而从三方面综合介绍了基于错误模型的设计验证技术的研究现状，以及我们在这方面开展的部分工作。本文首先阐述了设计错误模型的研究现状，重点介绍了差异测试的思想以及缺项错误模型及其测试方法；其次介绍了错误注入系统 ErrorInjector，该系统能够支持种类丰富的设计错误模型，而且具有良好的可扩展接口，是一个对于设计错误模型研究非常有益的基础平台；最后介绍了基于错误屏蔽概率的静态可观测性量化分析方法，以及在此基础上的根据低观测根源选择内部观测点的方法，为从设计错误所引发的效果角度研究可验证性设计技术提供了一个实例。

关键词：芯片设计、模拟验证、错误模型、错误注入、可验证性

随着集成电路工艺的不断发展，芯片设计的规模越来越大，复杂程度越来越高，对设计的功能正确性以及性能、功耗、可靠性等都提出了更高的要求。其中，功能正确性是芯片设计最基本的要求，需要通过设计验证（design verification）技术解决，这就使设计验证技术成为芯片设计的重要支撑技术。

在集成电路发展的早期阶段，设计验证工作通常是通过一些定制的甚至是手工的方式来完成，例如，通过设计人员手写测试用例（test case），或者通过运行典型应用程序，来检验芯片设计的正确性。随着设计规模的持续膨胀，设计验证方面的研究得到了学术界越来越多的关注，尤其是近 10 年以来，已经成为学术界关注的一个热点。这主要是由于已有设计验证技术的处理能力已经无法满足设计规模发展的需要。根据美国 Collett 市场调研公司的报告：2002 年，功能错误已经成为芯片重流片的首要原因，而这一因素的比重在 2004 年继续呈现增长的趋势。

影响设计验证技术处理能力最直接的因素是芯片设计的功能复杂度。然而这一因素很难通过某种指标来量化度量。常用的度量设计规模的晶体管数目、逻辑门数目等等，都无法全面地反映芯片设计的功能复杂度。2004 年度国际半导体技术发展报告（ITRS）指出："设计规模随着摩尔定律成指数级增长。如果采用系统中需要验证的不同状态的数目来衡量其功能复杂度，那么在最坏情况下，功能复杂度随设计的规模也成指数级增长。这导致了一个双指数级的爆炸[1]"。可见，设计验证所面临的问题规模的增长趋势比设计规模的增长要快得多。除了对设计的状态进行检验，设计验证技术常常还需要在状态序列上进行推理，这种情况下的问题规模膨胀得更快。根据 ITRS2008 的预测，2019 年芯片上集成的晶体管数量将达到数十亿数量级。设计规模的提高将使得设计验证技术面临极大的挑战。因此，ITRS2008 指出，如果验证技术没有重大突破，将使得半导体工业的后续发展呈现止步趋势[1]。

现有的设计验证技术，从方法角度来看，通常可以划分为模拟验证（simulation-based verification）和形式验证（formal verification）两个大类。通俗的讲，模拟验证是通过施加输入向量进而分析芯片设计在模拟器（simulator）上的行为来发现设计错误，而形式验证是利用数学方法来严格地证明一个芯片设计满足所给定的需求（即规范）。二者结合之后又产生了半形式化验证技术。

[1] 本文的部分内容曾出现于参考文献[29]和[30]之中。

在这两大类技术中，模拟验证是工业界一直采用的、也是占主体地位的验证技术。形式验证能够隐式地穷举整个电路的输入变量空间，证明电路设计的功能正确性。虽然其重要性日益上升，但是由于形式验证技术对于设计规模更加敏感，目前尚未成为工业界设计验证的主要方法。本文将分析模拟验证技术目前所面临的挑战，进而介绍基于错误模型的设计验证技术。这是一个有望取得突破性进展的技术方向。

1　模拟验证所面临的挑战

在这一节中，我们首先简要介绍模拟验证技术，然后分析其所面临的挑战，并阐述研究基于错误模型的设计验证技术的意义。

1.1 模拟验证简介

模拟验证之所以成为业界常用的、也是最主要的验证技术，主要是由于其具有强大的可扩展性。一般来讲，任何规模的设计总是可以通过模拟的方法来检验的。开发模拟验证环境（testbench）所用的语言经历了几番更新换代--从直接采用硬件描述语言（hardware description language，HDL），到采用面向对象的语言（例如 C++），再到专门的验证语言（例如 SystemVerilog 语言和 e 语言）。专门的验证语言提高了设计验证的效率，使得验证工程师可以方便快捷地开发出功能强大的验证环境。模拟验证中的典型验证环境如图 1 虚线中的部分所示。图中虚线内是验证环境的组成模块，虚线外部的结点表示与模拟验证过程有关的文档，图中连线表示信息或数据的流向关系。

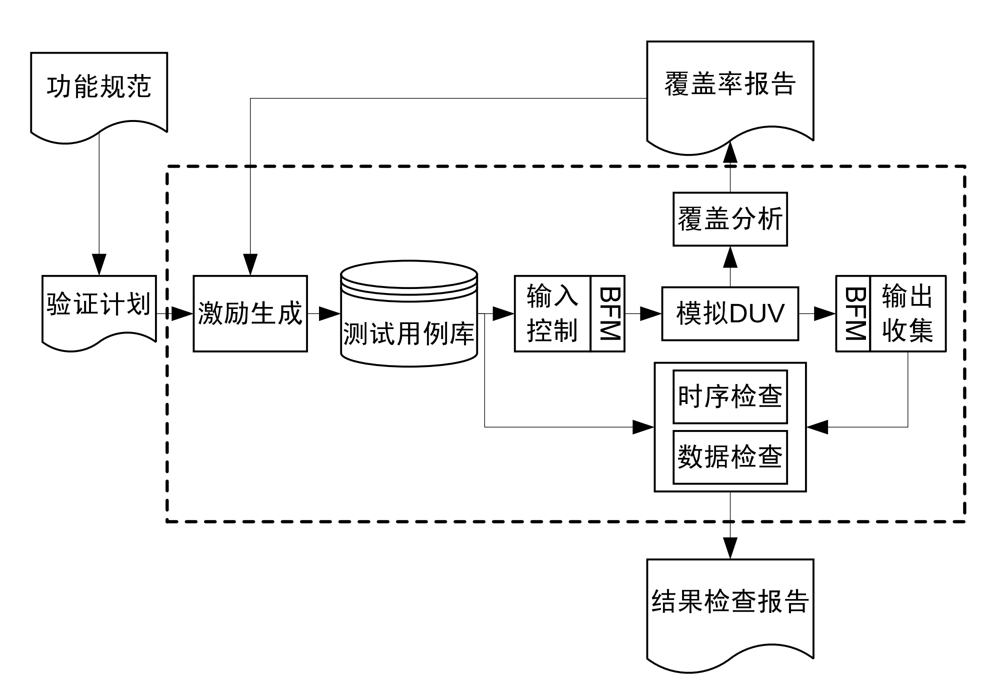


图1.　典型的模拟验证环境

在模拟验证过程中，验证工程师首先需要理解功能规范并制定出相应的验证计划，然后依照验证计划开发测试用例。除了定制一些测试用例之外，经常采用的技术还有基于约束的随机激励生成（constraint-based random vector generation）。为了提高验证环境的开发效率和验证环境中模块的可重用性，验证语言通常采用类似面向对象语言的方式来描述测试用例。因此在向被验证设计（design under verification，DUV），尤其是寄存器传输级（register-transfer level，RTL）或门级的被验证设计施加测试用例时，需要将较抽象的测试用例映射成二进制

向量[2]，必要时还需要通过总线功能模型（bus functional model，BFM）[2]按照被验证设计所需要的时序关系来施加二进制向量，这就是图 1 中输入控制部分需要实现的功能。输出收集部分的功能则是输入控制部分的功能的逆过程。对于模拟过程中收集到的数据，一方面需要进行结果检查，包括时序检查和数据检查，形成结果检查报告以反映模拟验证过程中是否发现了设计错误；另一方面需要进行覆盖率分析，形成覆盖率报告以评估模拟验证的充分程度。对于设计验证环境中各部分的详细介绍，可参见文献[3]。

1.2 纳米时期模拟验证所面临的挑战

正如前面所解释的那样，设计验证的问题规模随着摩尔定律成双指数级增长，而当集成电路的工艺尺寸发展到纳米级，芯片设计的规模发展到数 10 亿晶体管级，设计验证的问题规模将是现有技术难以应对的。

根据 1.1 节的介绍，模拟验证中的主要环节包括：(1)激励生成、(2)模拟、(3)结果检查、(4)覆盖率评估。其中结果检查环节通常采用断言的方式或者与参考模型比较的方式来实施，这方面的技术相对成熟。在其它的三个环节中，提高模拟速度有助于在更短时间内验证更大功能空间。例如采用硬件加速的方法可以将模拟的速度提高几个数量级，但是这仍然跟不上设计的功能复杂度的增长。因此，模拟验证研究的重点在于激励生成和覆盖率评估这两个环节。

功能覆盖率（functional coverage）评估是业界最常用也是最重要的度量验证进度的方法。然而，定义高质量的功能覆盖点需要工程师具有大量的专家知识和相关经验，这也是功能覆盖率评估主要的不足之处。也就是说，功能覆盖率的度量效果依赖于所定义的功能覆盖点的完备性，而这种完备性目前尚没有方法能够检查。在模拟验证的过程中运用覆盖率评估方法，可以分析出覆盖率漏洞，这些信息对于激励生成具有重要的指导意义。如果功能覆盖点的定义不完善，那么在缺乏有效覆盖率信息指导的情况下想要通过激励生成的效果来弥补是非常困难的。

模拟验证中的激励生成通常在寄存器传输级及其以上的层面上完成。这种情况下激励生成本身是很困难的，主要有三个方面的原因：(1)待验证的属性常常涉及到多个时钟周期，问题空间庞大；(2)相应地，设计验证中的激励生成常常是面向时序电路的，而不像电路测试中那样（通过扫描链等可测试性设计技术的转化）主要处理组合逻辑；(3) 寄存器传输级及其以上的设计中常常包含位向量（bit-vector）或者说字变量（word-level variable），其取值范围相比门级设计中的布尔量要大得多，相应的运算也更加复杂。这三方面的因素导致设计验证领域的激励生成通常不采用确定性的算法，而是大多采用一些随机算法[4]，或者基于模拟的方法[5][6]。如果功能覆盖点的定义不完善，那么通过基于随机或者模拟的激励生成方法很难在短时间内发现相关逻辑中的设计错误。例如，英特尔宣称奔腾（Pentium）芯片的浮点除法错误对一个普通制表软件用户而言，每 27,000 年才遇到一次[7]。

综合上述，模拟验证的激励生成非常困难，因此需要覆盖率信息的引导，而功能覆盖率一方面需要太多的人工参与，另一方面其目标空间过于庞大，与设计规模成指数关系，因此现有的围绕功能覆盖率的模拟验证技术已经很难满足芯片设计未来发展的需要。

1.3 开展错误模型相关研究的意义

发现设计错误是模拟验证最直接的目标。在集成电路测试领域，由于固定型故障

[2] 在后文中，当不强调功能场景时，通常不讲“测试用例”而直接讲“向量”、“输入向量”或者“激励”，这些均指施加在被验证设计上的一个时钟周期的输入数据。

（stuck-at fault）模型的成功提出，使得测试生成、可测试性分析、故障覆盖率评估等工作的自动化程度非常高，测试的完备性也有据可依。类似地，如果能够提出高质量的设计错误模型，有望极大地促进模拟验证的相关技术研究。类似的观点在 ITRS2007 也有所反映[1]。

首先，设计错误模型研究有望降低设计验证技术的问题规模。如前所述，从功能的角度进行设计验证，其问题规模随设计规模呈指数增长。而如果从设计错误的角度进行设计验证，由于很多错误模型都是从分析某一类设计错误所带来的代码改变来建模的，因此问题规模基本上与设计规模（代码）成同数量级。

其次，设计错误模型研究与现有围绕功能覆盖率的模拟验证技术可以形成紧密的互补。一方面，基于错误模型的设计验证离不开对基本功能的评估，毕竟设计验证的最终目的是要保证设计满足其功能规范；另一方面，现有的围绕功能覆盖率的模拟验证技术由于问题规模过于庞大而难以对各种边界情况实现完备的覆盖，而基于错误模型的技术则可以从代码中潜在错误的角度来弥补这一点，形成二者紧密互补、互相促进的局面。虽然现有的语句覆盖率[6]、分支覆盖率[8]等评估方法也是从代码角度来评估验证完备性的，但是这些覆盖率评估方法通常是采用某种启发式思想，与设计验证的效果之间缺乏直接的联系，因此对其评估效果一直以来缺乏充分的证明，通常只能将 100%的代码覆盖率作为模拟验证需要满足的基本要求。基于错误模型的设计验证技术在这方面则占有先天优势。换句话说，发现设计错误是模拟验证体现其效果的最直接的方式，当对某种错误模型达到了 100%的覆盖之后，我们通常可以说设计之中没有该类设计错误了。

简而言之，基于错误模型的设计验证技术是主流模拟验证技术应对设计规模发展带来的挑战的有效方法，受到业界广泛关注。

2　设计错误模型

在芯片设计流程中，多种多样的原因都可能导致功能错误。例如，由于版本管理不善导致的错误、由于更改功能规范而导致的实现与规范之间的不匹配、由于接口不兼容而导致的连通性问题，等等。本文并不打算探讨所有可能的设计错误，而仅限于由设计人员所带来的功能错误。

2.1 错误模型相关研究

表 1. 错误类型相关研究分类

类型	特点	代表性工作	
显式错误模型	通过所引起的设计代码的改变来描述设计错误	软件层	[9]
		RTL	[13]、[14]、[16]、[24]
		门级	[15]
抽象错误模型	在电路设计的抽象模型上提炼错误模型	[17]、[18]	
隐式错误模型	从错误所引发的效果来分析设计错误的特性	[19]、[20]、[21]	

如表 1 所示，从建模角度来看，错误模型方面的研究可以划分为三类。

第一类是显式错误模型，即通过所引起的设计代码的改变来描述设计错误。差异测试（mutation testing）是这一类错误模型研究中的典型方法。差异测试最早是由软件测试领域的学者在文献[9]中提出的。其主要思想是对原始设计进行微小变动以产生各种变异体（mutant），然后生成能够区分原始设计与其变异体的测试向量。若原始设计在测试向量下行为正确，则其中不包含与相应变异体所对应的设计错误。

差异测试中的错误模型是从常见的设计错误中提炼出来的，例如将逻辑“与”运算（&）误写为逻辑“或”运算（|），或者将关系运算中的大于等于（>=）误写为大于（>）。 每个变异体含有唯一的模型化错误，这一方面是为了便于计算错误覆盖率，另一方面是基于两个假设——合格程序员假设和错误的耦合效应假设。“合格程序员假设”指的是，一个合格的程序员编写出来的程序总是接近于正确的设计，如果出错也只是一些相对简单的错误。“错误的耦合效应假设”是指，针对简单错误而生成的激励集合也能够发现那些由简单错误组成的复杂错误。这一假设于 1978 年在文献[9]中被提出，软件测试领域的学者从实验的角度[10]和理论分析的角度[11]分别对这一假设进行了研究。

文献[12]中报告了针对 Fortran 程序的自动化的差异测试系统 Mothra。由于软件程序过于复杂，而且变异体数量非常庞大（例如，Mothra 曾对一个 27 行的程序产生了 970 个变异体[12]），这导致差异测试的开销过大，因此未在软件测试中得到广泛应用。尽管如此，差异测试为设计错误模型方面的研究奠定了基础，其思想也已经被借鉴到硬件的设计验证技术之中[13]。

巴克莱（D. Barclay）等人针对 VHDL[3]代码中的 `if-then-else`、`case-when`，和信号赋值结构提出了控制错误（control fault）模型。这些条件结构中的谓词在验证过程中被要求固定为 1 和固定为 0，而赋值操作被要求开路（open）以检测死子句（dead-clause）[14]。文献[15]针对门级组合电路定义了四类简单的错误模型，即门类型错误、门数目错误、输入数目错误和输入信号错误，并将错误模型映射到 SSL 故障（single-stuck line fault，线的单固定型故障）以借助测试自动生成（Automatic Test Pattern Generation，ATPG）工具产生激励。文献[16]则提出了较高层面上的设计错误模型，包括总线顺序错误、总线源错误，等等，并针对电路的寄存器传输级设计手动产生测试集。

第二类是抽象错误模型。其主要思想是抽象出电路设计的某种模型，在其上分析提炼出功能错误模型。例如，闵应骅等人提出了一种寄存器传输级模型，在此基础上提出了多种错误模型，包括译码故障、数据存储和传输故障等等[17]。沈理等人针对微处理器设计也提出了相应的功能错误模型[18]。

第三类是隐式的错误模型，从错误所引发效果的角度来分析设计错误的特性。例如，文献[19]通过将电路设计方案表达成多项式的形式，借助多项式的零点理论进行分析设计错误。文献[20]提出基于屏蔽值集合（masked value set， MVS）的概率计算方法，来评估基于可观测性的语句覆盖率。我们所提出的静态可观测性分析方法也是从所引起的变量取值变化角度来评估设计错误的影响[21]。

以上设计错误模型方面的研究主要集中在二十世纪的八十、九十年代，近十余年很少有重大进展。近期研究主要是针对一些特定的设计错误建立模型，例如，北京大学的学者针对多时钟域的 SoC[4]设计中的亚稳态效应（effect of metastability）提出跨时钟域错误模型[22][20]。随着设计规模不断膨胀，电路设计的发展对验证技术提出了紧迫的需求。在这样的背景下，基于错误模型的设计验证有望与现有主流的模拟验证技术形成紧密互补，因此具有令人期待的发展前景。

2.2 缺项错误模型

我们在实际芯片的设计验证过程中发现有表达式中子句缺失的设计错误。这类设计错误在其它的芯片设计中也同样存在。加州大学洛杉矶分校的学者分析了英特尔奔腾 II(Pentium

[3] 硬件描述语言

[4] System on Chip，片上系统

II)微处理器的功能缺陷列表（errata），发现英特尔在 1997 年 5 月至 1999 年 4 月所报告的奔腾 II 微处理器的 73 个功能缺陷之中，大部分是控制错误。这些控制错误又可以大致分为两类：控制功能的缺失或控制功能出错[23]。我们发现表达式子句缺失的设计错误，与文献[23]中的基本认识不谋而合，并且对其进行了新的拓展——除了控制逻辑之外，在赋值语句的右值中也可能有表达式子句缺失的设计错误。但是已有错误模型和测试方法尚不能处理这种情况。我们在文献[24]中结合实际工程项目中的设计错误分析提出了一种新的设计错误模型—缺项错误（item-missing error，IME）模型，可以处理此类表达式中的子句缺失错误。本文中我们仅以描述性的语言来举例说明缺项错误模型及其测试方法，更详尽更形式化的阐述可参考文献[24]。

我们所提出的缺项错误模型可以给硬件设计中由于子句缺失所导致的错误建模，包括在连续性赋值、非阻塞性赋值、阻塞性赋值等语句右端表达式缺失子句，以及在 if 语句、case 语句、条件操作符（即问号操作符）等结构中的条件表达式缺失子句。

下面结合实际项目中的设计错误来说明缺项错误模型的实际意义。

图 2(a)所示是含有设计错误的代码，图 2(b)是其修复后的代码。这段代码是 AMBA AHB 主设备（master）接口的复位（`reset`）逻辑中的一个片段。除了主设备接口逻辑自身的初始化（`initialization`）操作之外，还有一种情况下也需要对主设备接口逻辑进行复位，即当从设备（slave）返回一个 ERROR（出错）类型的响应之时，主设备接口逻辑也需要进行复位。此错误可以通过 IME 模型来建模。

```
if (!hresetn)
  buffer2[294:293]<=2'b0;
```

(a). 含有设计错误的代码

```
if ((!hresetn)|bus_error)
  buffer2[294:293]<=2'b0;
```

(b). 错误被修复之后的代码

图2. 可用缺项错误建模的简单的设计错误

根据 AMBA AHB 协议规范[25]，一个 ERROR 类型的响应需要至少两个时钟周期来表示：在倒数第二个时钟周期，从设备将 HRESP[1:0]信号设置为 2’b01 并同时将 HREADY 信号拉低，而在最后一个时钟周期，从设备保持 HRESP[1:0]信号不变同时将 HREADY 信号拉高。可见要识别 ERROR 类型的响应需要相对复杂的逻辑，因此在修复缺项错误时可能需要将一些信号进行运算，而并非简单添加一个内部信号或者输入信号（图 2(b)中添加的 `bus_error` 信号是识别 ERROR 类型响应对应的两时钟周期模式而产生的）。这样的设计错误是无法通过变异测试技术发现的。换句话说，变异测试是在已有代码基础上进行变异的，它并不能处理那些由于缺失而没有在代码中出现的子句。因此缺项错误需要通过新的测试方法来检测。

我们提出一种基于约束的随机激励生成方法来检测缺项错误。

以图 3 中代码为例。图 3(a)第 1 行 **if** 语句中的表达式“a&b”有一个缺项错误，其正确形式是如图 3(b)所示的“a&b&c”。当 a=1，b=1，c=0 之时，图 3(a)中的错误代码将执行第 2 行的 **if** 分支，而图 3(b)中的正确代码将执行第 4 行的 **else** 分支。若此时“`input1|input2`”的值不等于“`input1|input2`”，那么这个缺项错误就能够被发现。这个例子带来的启示是，如果在激励生成过程中约束已有的项取其非控制值（关于非控制值的定义可参见文献[24]），就有

```
1. if (a&b)
2.   p=input1|input2;
3. else
4.   p=input1&input2;
```

(a) 含有缺项错误的代码

```
1. if (a&b&c)
2.   p=input1|input2;
3. else
4.   p=input1&input2;
```

(b) 正确代码

图3. 缺项错误示例

可能将所缺失的项的效果暴露出来。

在实际的设计验证过程中，无法提前得知所缺失的项是什么。例如在图 3(a)所示例子中，其正确代码可能是“a&b&c”，也可能是“a&b&d”或者“a&b&(!e)”。因此，我们所提出的缺项错误模型测试方法，其目标并不在于如何找到正确代码，而是在于提高发现此类设计错误的概率。为了使所缺失的项的作用尽可能暴露出来，就需要将已有的项的效果尽可能隐蔽。

显然，对于某个表达式中的单缺项错误，在其它条件相同的情况下，相对于不满足非控制值约束的激励而言，那些使得已有项满足非控制值约束的激励发现该缺项错误的概率更大。如果某个已有的子表达式取控制值，那么无论所缺失项的效果有没有被激发，整个表达式的取值都取决于这个子表达式的控制值。这样的激励是不可能检测出该缺项错误的。

基于业界常用的基于约束的随机验证技术，我们提出**针对缺项错误模型的测试方法**，其流程如图 4 所示。其中的核心步骤包括：第一步，选择一个潜在的缺项错误；第二步，得到相对于该缺项错误的其它子表达式非控制值约束，将此约束追加到验证环境的功能约束之上，产生满足这些约束的随机向量，对该缺项错误进行若干次测试；第三步，如果还有其它的潜在缺项错误则转回第一步，否则结束测试。

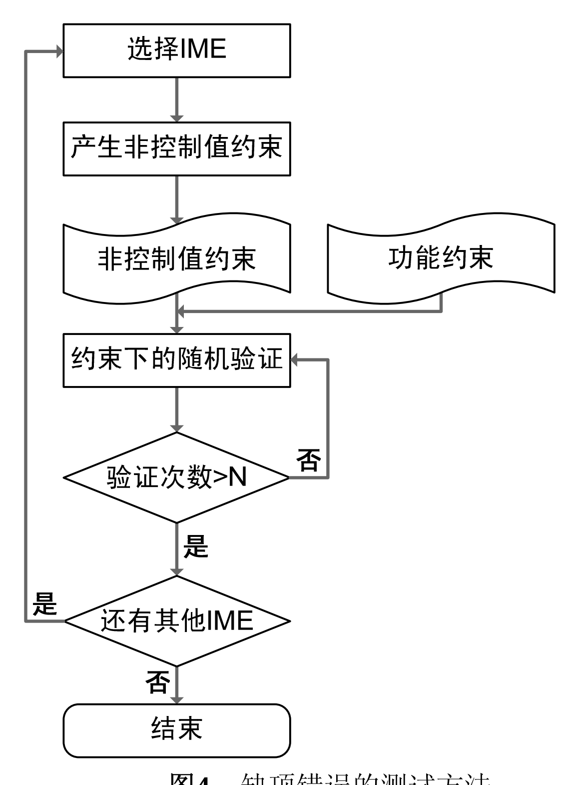


图4. 缺项错误的测试方法

如图 5 所示，对于实际的嵌入式处理器设计的实验数据表明，对于缺项错误而言，在添加 IME 结构约束情况下平均被发现概率，是不添加 IME 结构约束情况下平均被发现概率的数倍，最高达到了六倍以上。这说明从缺项错误模型提炼出的结构信息对于发现此类设计错误是非常有用的，能够极大提高功能验证的效率。详细的实验数据和分析可参考文献[24]。

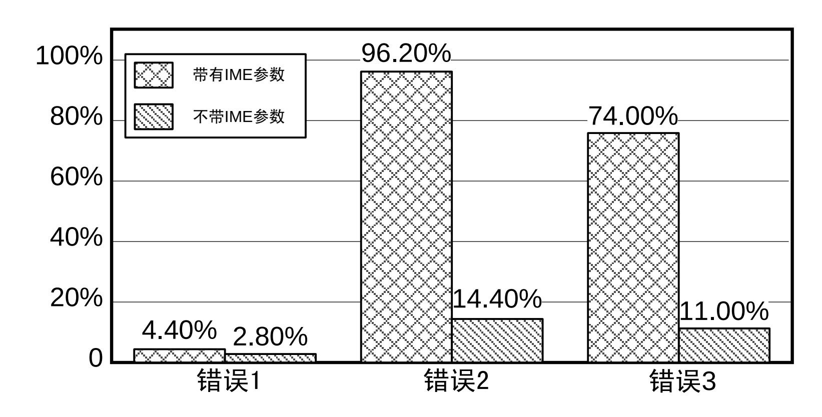


图5. 缺项错误模型对于提高该类错误被发现概率的作用

3　设计错误注入平台 ErrorInjector

由于目前尚未有得到广泛承认的设计错误模型，相应地也没有高效的**设计错误模拟器**，

因此，分析和比较各种设计错误模型的优劣、评估设计错误覆盖率的基本方法就是进行**错误注入**（error injection）和**逻辑模拟**（logic simulation）。据我们了解，目前尚没有开源的针对硬件设计的错误注入系统，因而相关的研究团队需要自行开发，造成时间等资源的重复消耗。为此，我们设计并实现了一个设计错误注入平台——ErrorInjector。

在设计错误模型相关研究中，一个良好的设计错误注入系统的很重要的特性就是能够支持丰富的错误模型种类。如果系统能够支持灵活的错误注入位置则更好，这样可以在整个设计中进行随机的注入或者在关注的区域进行集中的错误注入。ErrorInjector 正是具有这样的特征。具体而言，ErrorInjector 的主要特点在于：

1. 借助编译器提取设计信息，可支持多种错误类型。该系统支持 Verilog 语言，利用开源编译器 IcarusVerilog[26]进行设计信息的提取。
2. 支持错误注入位置的随机选取方式和等间距选取方式。该系统设置了预处理步骤，通过预处理统计出设计中各种错误模型对应的注入位置的数目，提供给用户以便描述错误注入的配置信息。
3. 基于动态链接库技术实现错误模型的可扩展接口，使系统具有良好的可扩展性。用户仅需构造出接口函数（即与具体错误模型相关的识别函数和注入函数）的内容并将其编译为动态链接库文件，就可以将新的错误模型融合到 ErrorInjector 系统当中。这是一种友好的扩展方式，使得用户无需了解 ErrorInjector 系统的所有实现细节就可以扩展所需的错误模型，极大减少了用户的工作量，提高了系统的易用性和可扩展性。

ErrorInjector 系统的错误注入流程如图 6 所示。

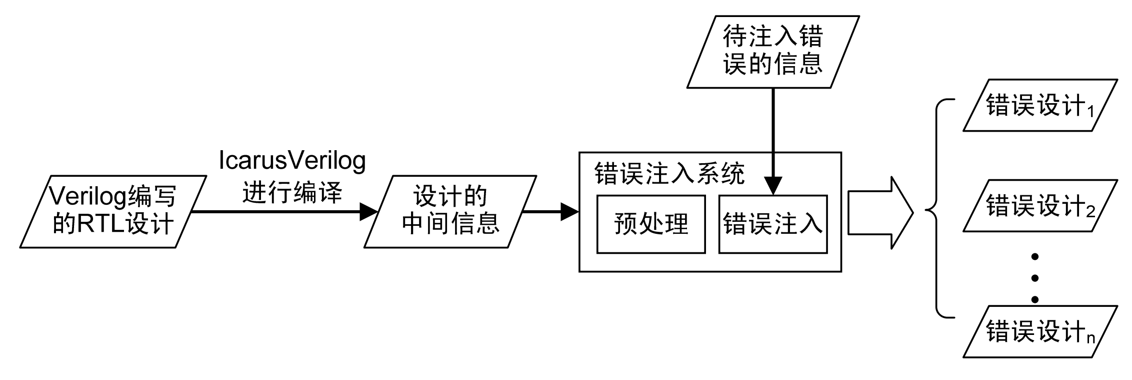


图6. 错误注入流程

ErrorInjector 系统需要两个输入：第一个输入是被验证设计，即采用 Verilog 语言实现的硬件设计；第二个输入是验证人员指定的待注入错误的信息，包括准备注入的错误模型、错误个数和错误位置等。输出则为多个带有单错误的设计文件，每个输出设计都是符合语法的。目前的系统是在 Linux 平台下用 C++语言实现的。其运行界面如图 7 所示。用户可以通过图形界面了解芯片设计中所有可能的错误信息，并对具体注入的错误类型、数目、位置等进行相应的配置。

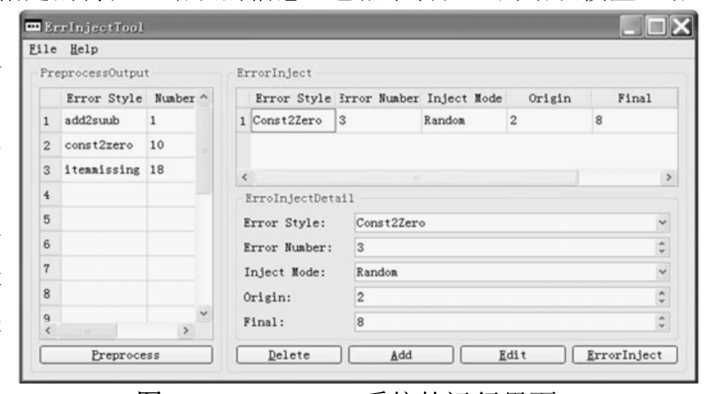


图7. ErrorInjector 系统的运行界面

基于该平台可以开展以下两方面的研究：

1. 分析和比较各种设计错误模型的优劣。对于在实际芯片验证中已经发现某些设计错误的激励集合而言，通过进行错误注入并利用该激励集合进行模拟，能够分析各种错误模型对实际的设计错误的建模能力，进而提炼出具有代表性的错误模型以指导设计验证技术。

2. 评估错误覆盖率以指导激励生成。通过对芯片设计进行单错误注入，可以得到多个带有设计错误而语法正确的设计。再借助逻辑模拟器对这些设计分别进行模拟验证，或者借助形式化工具进行形式验证。最后，对验证的结果进行分析和统计，计算出被发现的设计错误数目与所有被注入的设计错误数目之比，即为**错误覆盖率**。分析所得到的覆盖率漏洞对于激励生成具有重要的指导意义。

4 基于隐式错误模型的可验证性设计技术

我们在研究错误模型的过程中深刻认识到，设计错误建模方面的研究困难很多，一个主要的原因是芯片设计厂商很少公开其项目中的设计错误详情。虽然英特尔等芯片厂商会提供其产品的功能缺陷列表（errata），但这通常是从用户的角度来描述，难以供设计验证研究使用。密歇根大学的学者系统地收集了学生在几个微处理器设计课程项目中的设计错误数据，并在文献[27]中进行了分析。文献[28]也报告了在对学生设计进行形式化验证的过程中所发现的设计错误。不过类似的公开数据还是少之又少。而且学生设计中所犯的错误并不能完全代表工业界设计中的错误。考虑到设计错误的显式建模研究所面临的客观困难，我们可以不必拘泥于特定的设计错误模型，而是从设计错误所引发的效应入手开展研究。对于可验证性设计尤其如此。

对于可验证性设计而言，通常不与某种特定的显式错误模型绑定，而是从错误所引发的效应入手分析验证的难易程度。我们从设计错误传播角度研究量化分析方法，提出了基于错误屏蔽概率（error-masking probability，EMP）的静态可观测性（Static. OBServability，SOBS）分析方法。基于该方法得到的设计中内部信号的静态可观测性量化分析值，我们提出了内部观测点选择算法。该算法能够在模拟之前定位出造成局部设计难观测的源头。将这样的源头作为内部观测点，可以提高设计验证发现错误的能力。

4.1 研究静态的观测点选择方法的动机

图 8 是静态可观测性分析方法作用的示意图。图中横轴表示模拟的时间，纵轴表示验证的效果。常用的方法就是通过覆盖率来体现验证的效果。图中有两条覆盖率曲线。靠上方的一条相对平滑的曲线表示采用静态分析方法添加内部观测点时的覆盖率增长情况。靠下方的阶段式增长的曲线表示采用传统方法添加内部观测点时的覆盖率增长情况。

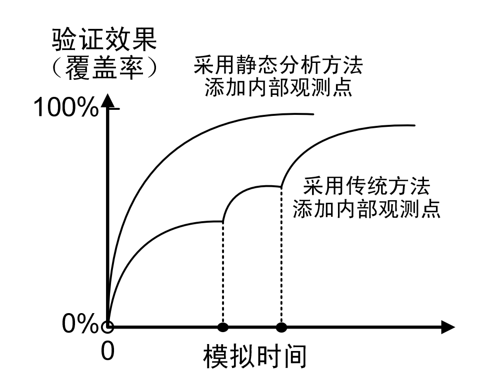


图8. 静态可观测性分析意义的示意图

传统方法通常是先进行一段模拟，在模拟过程中没有覆盖到的部分被认为是难观测的。由验证人员为其增加内部观测点，再进行模拟。如此反复。如图 8 中的示意，横轴上的两个实心圆点表示采用传统方法添加内部观测点

的时机。由于是在模拟过程中添加的，新增的内部观测点就只能在其后的模拟中起作用，因此覆盖率曲线呈阶段性增长。而且，对于模拟过程中没有覆盖到的为数不少的区域，选择哪些作为内部观测点要完全依赖于人力来决定，因此缺乏稳定性，导致覆盖率增长相对缓慢。

我们针对这个问题提出的静态可观测性分析方法不需要模拟数据来辅助分析，因此可以在模拟开始之前就报告出造成部分区域难观测的源头，提示验证人员将其增加为内部观测点。这样可以避免传统方法依赖人力而可能造成的盲目性，新增的内部观测点在模拟开始的时候就可以起作用，使覆盖率曲线平滑快速增长。

4.2 根据低观测根源选择内部观测点的方法

本文讨论的**设计验证中的可观测性**是指观测到设计中某个信号上的错误效果的难易程度。**静态可观测性**（SOBS）是用来评价硬件设计中某个内部信号可观测性的具体量化指标。对于某个内部信号而言，其 SOBS 值表示该信号上的设计错误被屏蔽的可能性大小。

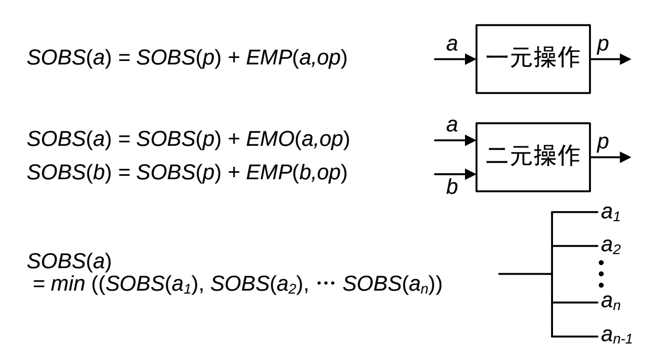


图9. SOBS 计算方法

我们从操作数取值的角度来估计设计错误传播的难易程度。无论是何种形式的设计错误，其效果通常反映为对设计中的信号值的影响。由于不同的测试用例会执行设计中的不同路径、遇到不同的操作，而信号值在不同寄存器传输级操作上传播的难度不同，这导致了不同测试用例下设计错误的传播效果也不同。

寄存器传输级典型字操作的错误屏蔽概率计算方法可参见文献[29]，这里不做详细推导。在错误屏蔽概率公式基础上，我们可以分析硬件描述语言设计中内部信号的静态可观测性。首先，构造硬件描述语言设计的控制/数据流图（Control Data Flow Graph，CDFG）。然后，从观测点出发沿着控制/数据流图中的路径逆向分析，同时根据图 9 所示的公式计算内部信号的 SOBS 值。验证过程中的观测点（例如，原始输出变量）的 SOBS 值设为 0。当沿着控制/数据流图中的多条路径均可以观测内部信号 a，则 a 的 SOBS 值取各条路径上 SOBS 值中的最小值。对于分支语句，其条件表达式的 SOBS 值取其分支中被赋值的各个信号的 SOBS 值中最小值。

由此可知，1).SOBS 值越大，表示信号越难以被观测；2).从观测点开始逆向查看一条路径上的 SOBS 值，其值具有单调递增的特性。基于这两个特性，我们提出了低观测源分析方法。此方法采用打分的方式来筛选内部观测点，即利用 SOBS 值计算每个内部信号的 `IOS_score`，最终 `IOS_score` 分值最高的内部信号就是理想的内部观测点。

此方法的主要步骤有如下 3 步。

1. 遍历电路设计所对应的控制-数据流图，求出每个内部信号的 SOBS 值。如果某个信号的 SOBS 值大于所设定的阈值 TSOBS，则该信号被视为一个**难观测信号**。

2. 对每一个难观测信号 a，沿着其最容易传播的一条路径 l（即在该路径上计算得到的 a 的 SOBS 值最小）分析。因此，**如果路径 l 上的两个相继信号的 SOBS 值之差足够大，则表明这两个信号之间的操作 op 是错误效果在路径 l 上传播的一处重要障碍**，故将这两个相继信号中对应 op 操作数的那个信号的 `IOS_score` 递

增。

3. IOS_socre 分值最高的那些内部信号就是内部观测点的理想候选。

对于一条路径 l，可以用路径上相继出现的信号序列的形式表示为 $s_1, s_2, \cdots s_{n-1}, s_n$，其中 s_n 是观测点。对路径 l 上的两个相继信号 s_i 和 s_{i+1} $(0 < i < n)$，通常情况下

$$SOBS(s_i) = SOBS(s_{i+1}) + EMP(s_i, op)$$

如果 s_i 和 s_{i+1} 的 SOBS 值之差大于所设定的阈值 TSOBS_diff，则表明由于 op 的存在使得信号 s_i 的可观测性相对其后继信号 s_{i+1} 差了很多。根据前面介绍的 SOBS 计算公式可知，一条路径上某个信号的 SOBS 值是其后继信号的 SOBS 值累积的结果。因此，信号 $s_1, s_2, \cdots s_{i-1}$ 的可观测性都受到 s_i 的影响。如果路径 l 上在 s_i 之前（即在信号 $s_1, s_2, \cdots s_{i-1}$ 之中）有难观测信号，则 s_i 是其难观测的一个原因。最终，如果某个信号 a 的 IOS_score 分值越高，表明电路中受信号 a 影响的难观测信号越多，因而将信号 a 作为内部观测点可以解决更多信号的观测难题，起到事半功倍的效果。关于更详尽的算法阐述和示例可参考文献[21][29]。

我们在 ITC-99 benchmark 电路上进行了实验与分析。对每一个设计，我们在同一个激励集合下比较其在 3 个方案下的错误覆盖率。这 3 个方案是指：(1)仅以输出变量作为观测点；(2)除了以输出变量作为观测点以外，从内部的难观测信号之中随机选择一些作为内部观测点；(3)除了以输出变量作为观测点以外，从低观测源分析方法所报告出信号中选择 IOS_score 分值最高的部分信号作为内部观测点。值得说明的是，为了尽可能实现方案 2 和方案 3 之间的公平比较，实验中选择内部观测点时尽量保证其总位数相当。

图 10 形象地说明了通过低观测源分析方法选择内部观测点的优势。图中的纵轴表示模型化的错误覆盖率，横轴表示电路设计。由于方案 2 中的内部观测点是在难观测信号集合中随机选择的，为了避免选择的偶然性，方案 2 被重复进行了三次，每次随机选择不同的难观测信号（由于 b14 中的难观测信号数目有限，所以方案 2 只能被重复 2 次）。图 10 中的方案 2-1、2-2、2-3 就是指对方案 2 的三次重复实验。

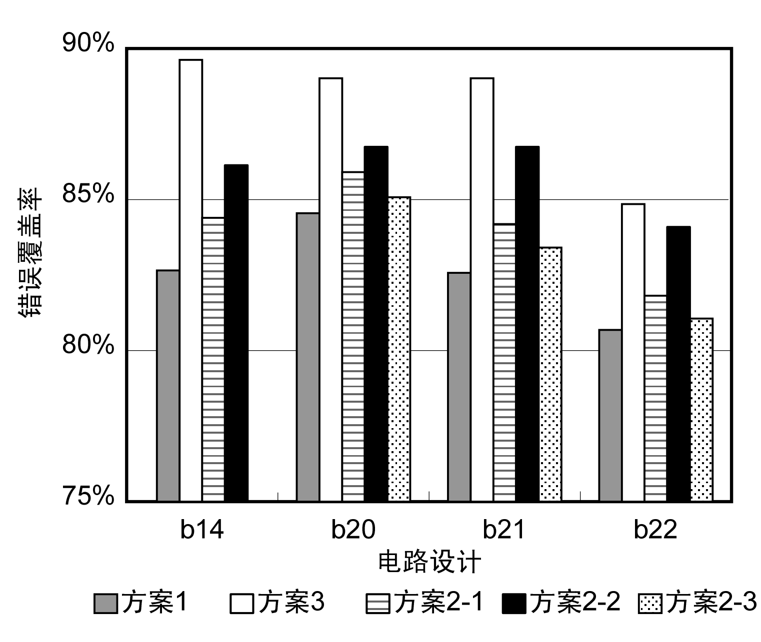


图10. 不同观测点下的错误覆盖率比较

由图 10 可知，尽管在方案 2 中随机选择不同的难观测信号作为内部观测点能够在不同程度上提高错误覆盖率，但是其提高的程度均未达到方案 3 的效果。由此可见，采用低观测源作为内部观测点，能够付出更小的代价解决更多内部信号的观测问题。

在时间开销方面，实验中的每个电路设计只需要几秒钟就可以分析完毕。而一些动态分析方法，例如文献[20]中的方法，则需要在功能模拟之后通过分析模拟结果来发现难观测信号，然后再进行模拟。由此可知，利用低观测源分析方法可以在功能模拟之前就分析并添加理想的内部观测点，因而可以节省验证开销。

5 总结

设计验证是芯片设计的重要支撑技术。通过分析业界主流的模拟验证技术在伴随设计规模发展中所面临的困境及其原因，我们认为基于错误模型的设计验证技术是有望取得突破性进展的技术方向。

本文阐述了设计错误模型的研究现状。其中，显式错误模型是一类备受关注的研究领域，这方面的研究以差异测试为代表。对于表达式中子句缺失的设计错误，已有的错误模型无法表示，我们则提出了缺项错误模型及其测试方法，能够极大提高该类设计错误被发现的概率。到目前为止，在设计验证领域尚没有得到广泛接受的错误模型，在这样的背景下研究设计错误模型相关技术离不开错误注入平台。本文介绍了我们所开发的错误注入系统 ErrorInjector。该系统基于编译器构建，能够支持丰富的设计错误模型，而且具有基于动态链接库的良好的可扩展接口，用户可以方便地开发自定义的错误模型。该系统对于设计错误模型研究非常有益。除了显式错误模型之外，从设计错误所引发的效果角度研究可验证性设计技术也是一个很好的思路。我们提出基于错误屏蔽概率的静态可观测性量化分析方法，在此基础上提出根据低观测根源选择内部观测点的方法，推动了这一领域的研究。

参考文献

[1] http://public.itrs.net

[2] Bergeron J, Writing testbenches using SystemVerilog, Springer, 2006

[3] Palnitkar S, Design Verification with e, Prentice Hall PTR, 2003.

[4] 易江芳，佟冬，程旭，“GATEST：使用遗传算法自动生成模拟矢量的验证平台”，北京大学学报（自然科学版），第 42 卷第 5 期，第 668-673 页，2006.

[5] I. Wagner, V. Bertacco, and T. Austin, “Microprocessor verification via feedback-adjusted Markov models,” IEEE Trans. on CAD, vol. 26, pp. 1126-1138, Jun. 2007.

[6] Wei Lu, Xiutao Yang, Tao Lv, Xiaowei Li, “An Efficient Evaluation and Vector Generation Method for Observability-Enhanced Statement Coverage”, Journal of Computer Science and Technology,Vol.20, No.6, Nov., 2005, pp.875-884

[7] http://www.intel.com/support/processors/pentium/fdiv/wp/

[8] Tao Lv, Lingyi Liu, Yang Zhao, Huawei Li, Xiaowei Li, "An observability branch coverage metric based on dynamic factored use-define chains," Proc. of IEEE 15th Asian Test Symposium, Fukuoka, Japan, Nov. 2006, pp.89-94.

[9] R. A. Demillo, R. J. Lipton, F. G. Sayward, “Hints on test data selection: help for the practicing programmer,” IEEE Computer, pp.34-41, April 1978.

[10] A. J. Offutt, “Investigations of the software testing coupling effect,” ACM Transactions on Software Engineering Methodology, 1992, vol. 1, pp. 3-18.

[11] K. S. H. T. Wah, “Fault coupling in finite bijective functions,” The Journal of Software Testing, Verification, and Reliability, 1995, vol. 5, pp. 3-47.

[12] K. N. King, A. J. Offutt, “A Fortran language system for mutation-based software testing,” Software-Practice & Experience, Vol.21, No.7, pp.685-718, 1991.

[13] G. Al Hayek, C. Robach, “From specification validation to hardware testing: a unified method,” Proc. of International Test Conference, pp.885-893, 1996.

[14] D. Barclay, J. Armstrong, “A heuristic chip-level test generation algorithm,” Proc. of Design Automation Conference, pp.257-262, 1986.

[15] H. AL-Asaad, J. P. Hayes, “Design verification via simulation and automatic test pattern generation,” Proc. of International Conference on Computer-aided Design, pp.174-180, 1995.

[16] D. Van Campenhout, H. AL-Asaad, J. P. Hayes, T. Mudge, R. B. Brown, “High-level design verification of microprocessors via error modeling,” ACM Trans. on Design Automation of Electronic Systems, Vol.3, No.4, pp.581-599, Oct. 1998.

[17] Yinghua Min, Stephen Y. H. Su, “Testing functional faults in VLSI,” Proc. of Design Automation Conference, pp.384-392, 1982.

[18] Li Shen, Stephen Y. H. Su, “A functional testing method for microprocessors,” IEEE Trans. on Computers, Vol.37, No.10, pp.1288-1293, Oct. 1988.

[19] Katarzyna Radecka, Zeljko Zilic, “Design verification by test vectors and arithmetic transform universal test set,” IEEE Trans. on Computers, Vol. 53, No.5, pp. 628-640, 2004.

[20] T. Y. Jiang, C. N. J. Liu, and J. Y. Jou, "Observability analysis on HDL descriptions for effective functional validation," IEEE Trans. on CAD of Integrated Circuits and Systems, Vol.26, No.8, pp.1509-1521, August, 2007.

[21] Tao Lv, Huawei Li, Xiaowei Li, "Automatic selection of internal observation signals for design verification," Proc. of IEEE 27th VLSI Test Symposium, Santa Cruz, USA, May, 2009, pp. 203-208.

[22] Yi Feng, Zheng Zhou, Dong Tong, Xu Cheng, “Clock domain crossing fault model and coverage metric for validation of SoC design,” Proc. of the Conference on Design, Automation and Test in Europe, pp.1385-1390, 2007.

[23] A. Avizienis, Yutao He, “Microprocessor entomology: a taxonomy of design faults in COTS microprocessors,” Dependable Computing for Critical Applications 7, pp.3-23, Nov. 1999.

[24] Tao Lv, Tong Xu, Yang Zhao, Huawei Li, Xiaowei Li, "Bug analysis and corresponding error models in real designs," Proc. of IEEE 12th High Level Design Validation and Test Workshop, Irvine, USA, Nov. 2007, pp. 59-64.

[25] Advanced RISC Machines Ltd (ARM), AMBA AHB Specifica-tion (Revision 2.0), ARM IHI 0011A, http://www.arm.com /products/solutions/AMBA_Spec.html

[26] http://www.icarus.com/eda/verilog/

[27] D. Van Campenhout, T. Mudge, J. P. Hayes, “Collection and analysis of microprocessor design errors”, IEEE Design & Test of Computers, Vol.17, No.4, pp.51-60, Oct.-Dec. 2000.

[28] M. N. Velev, “Collection of high-level microprocessor bugs from formal verification of pipelined and superscalar designs,” Proc. of International Test Conference, pp.138-147, 2003.

[29] 吕涛，数字集成电路设计验证的量化评估方法研究，北京：中国科学院研究生院博士学位论文，2009 年.

[30] 王天成，数字集成电路中设计错误的注入方法和实现，中国科学院研究生院硕士学位论文，2009 年.

作者简介：

吕　涛：　中国科学院计算技术研究所系统结构重点实验室助理研究员，lvtao@ict.ac.cn
李华伟：　中国科学院计算技术研究所系统结构重点实验室研究员，博士生导师
李晓维：　中国科学院计算技术研究所研究员、博士生导师、中国科学院计算机系统结构重点实验室副主任，中国计算机学会理事、容错计算专业委员会主任、JCST 副主编